Oracle

Auditoría de contraseñas en Oracle Database

Realizado por: *Sebastián Guerrero Selma*
Revisor *Manual Palomo Duarte*

Índice

1. Introducción

Las contraseñas son la forma de autentificación de usuarios más habitual en los equipos informáticos. Para evitar accesos no autorizados, estas son normamente protegidas mediantes hashing utilizando un algoritmo para ello. Posteriormente serán almacenados en la correspondiente tabla en lugar de utilizar las claves en formato plano.

Durante la identificacion del usuario este proporcionará una contraseña que se codificará con el mismo algoritmo y se comprobará con el hash almacenado.

Un enfoque general sobre la protección de contraseñas podría basar en los principios fundamentales comentados en el artículo de Morris & Thompson [1]:

- **Uso de contraseñas complejas**: Uno de los aspectos mas debiles dentro de la seguridad, reside en la baja entropia utilizada para crear una contraseña. Los principales ataques para obtener en texto plano el valor de un hash consiste en el calculo de todas las posibles combinaciones de caracteres para una longitud dada (*brute force*) o en el uso de ataques de diccionario.

  Por lo que un sistema deberia forzar al usuario el cumplimiento de ciertas restricciones a la hora de generar una contraseña segura (*uso de minúsculas, mayúsculas, números, símbolos, etc*).

- **Uso de Salted Passwords**: Cada hash está asociado a un pequeño valor aleatorio llamado **salt**. Siendo uno de los parámetros que se pasan a las funciones que se encargan de la derivación de claves. El otro parametro suele ser normalmente una contraseña o **passphrase**. El valor devuelto por la funcion sera almacenado como la version codificada de la contraseña.

  Su utilizacion permite impedir el uso de ataques de diccionario que utilizan pre-cifrado de entradas de diccionario: **cada bit de salt utilizado duplica la cantidad de almacenamiento y calculo necesario**

  Por razones de seguridad, se mantiene oculto y separado de las tablas donde se almacenan las contraseñas. Esto proporciona cierta ventaja ante la situacion de un posible robo. Dado que para un atacante resultará más complicado probar simples contraseñas (*como palabras de diccionario comunes o nombres*) ante un hash que haya sido filtrado. Teniendo que calcular el hash devuelto por caracteres aleatorios, incrementando considerablemente el tiempo de cómputo.

- **Algoritmos hash de un solo sentido**: Al contrario de lo que sucede con la codificación o cifrado, la salida de estos algoritmos no puede ser descifrada con algoritmos complementarios para obtener la información original. No existe forma de revertir el proceso.

  Caracterizado por tomar un dato de tamaño variable (*en este caso la contraseña*) y generar un dato de tamaño fijo. Asegurando además, que si la contraseña es modificada en un único bit, se generará por la función hash un nuevo resultado completamente diferente al anterior.

2. Mecanismo de contraseñas usado por Oracle

Las contraseñas de las cuentas de usuario son almacenadas en la tabla SYS.USER$ empleando hashes de 8-byte conseguidos mediant un algoritmo de hashing sin documentar. En un estudio realizado por el **SANS Institute** y dirigido por Joshua Wright & Carlos Cid se recogieron una serie de puntos debiles que comprometian la protec-cion de contraseñas utilizadas en el mecanismo de autentificacion.

Dichos problemas incluian:

- Débil seleccion de 'Salted Passwords'.
- Pobre juego de caracteres permitidos.
- Débil algoritmo de hashing.

El conocimiento de esto por parte de un atacante permitiria obtener la contraseña en texto plano del hash almace-nado para un determinado usuario.

2.1. Débil seleccion de 'Salted Passwords'

Oracle utiliza una tecnica poco convencional para la obtención de los salt, anteponiendo el nombre de usuario a la contraseña antes de calcular el hash. Esto permite la posibilidad de obtener información sobre la contraseña de un usuario basandonos unicamente en su valor de hash y los credenciales conocidos para cualquier otro usuario.

```
SQL> CREATE user prueba identified by password;

User created.

SQL> CREATE user prueb identified by apassword;

User created.


SQL> SELECT username, password FROM dba_users WHERE username LIKE 'PRUEB\%';

USERNAME                       PASSWORD
------------------------------ ------------------------------
PRUEBA                         BBFF158371D315FC
PRUEB                          BBFF158371D315FC
```

Revisando el resultado de nuestra consulta, cualquier atacante podría tener fuertes evidencias de la relación exis-tente entre ambas contraseñas de usuario.

Además los valores generados para los salt no son aleatorios. Si bien es cierto que permite reducir la eficacia de un ataque de diccionario contra el hash de alguna contraseña larga. Pero eso no evita que un atacante pueda usar una tabla de posibles contraseñas para un usuario común (*por ejemplo SYSTEM*) y vaya probando en diferentes sistemas hasta dar con los resultados esperados.

2.2. Pobre juego de caracteres permitidos

Otro de los puntos débiles de Oracle es el pequeño abanico de caracteres que utiliza para obtener el hash de una contraseña. Antes de que este sea obtenido, los caracteres de la contraseña son transformados a mayúsculas, inde-pendientemente de cómo hayan sido introducidos estos por el usuario.

Este comportamiento puede observarse para una misma contraseña introducida de diversas formas en el sistema y comprobar los valores de sus respectivos hashes.

```
SQL> ALTER user prueba identified by "PasswoRd";

User altered.

SQL> SELECT username, password FROM dba_users WHERE username LIKE 'PRUEBA';

USERNAME                       PASSWORD
------------------------------ ------------------------------
PRUEBA                         BBFF158371D315FC

SQL> ALTER user prueba identified by "password";

User altered.

SQL> SELECT username, password FROM dba_users WHERE username LIKE 'PRUEBA';

USERNAME                       PASSWORD
------------------------------ ------------------------------
PRUEBA                         BBFF158371D315FC

SQL> ALTER user prueba identified by "PASSWORD";

User altered.

SQL> SELECT username, password FROM dba_users WHERE username LIKE 'PRUEBA';

USERNAME                       PASSWORD
------------------------------ ------------------------------
PRUEBA                         BBFF158371D315FC
```

Observando la salida devuelta por nuestras consultas comprobamos que el hash permanece constante ante las modificaciones realizadas a la contraseña. Esto supone una reducción de la entropía de nuestas claves (*por ejemplo, si usamos caracteres alfanuméricos, obtendremos un mecanismo que permite* 36^n *posibles combinaciones de longitud n, en lugar de* 62^n*, que serían las deseadas*).

Otro problema es la falsa sensación de seguridad que puede generar para una organización que ponga ciertas restricciones a la hora de generar sus claves, como la combinación de minúsculas y mayúsculas.

2.3. Débil algoritmo de hashing

El algoritmo utilizado para calcular los hashes no ha sido abiertamente documentado por Oracle, pero en 1993 apareció en el newsgroup de comp.database.oracle un mensaje donde se describía el algoritmo en detalle, reconociendo el uso de un **magic number** como parámetro de entrada.

El proceso puede ser descrito como sigue:

1. Concatena el nombre de usuario y la contraseña para producir una nueva cadena en texto plano.
2. Se convierte la cadena anterior en mayusculas.
3. Se contierte la cadena en texto plano al formato multi-byte, teniendo los caracteres ASCII el octeto superior establecido a 0x00.
4. Se cifra la cadena en texto plano (completando con 0 en caso de ser necesario) usando el algoritmo DES en modo CBC con el magic number 0x0123456789ABCDEF.
5. Nuevamente se vuelve a encriptar la cadena en texto plano con DES-CBC, pero utilizando esta vez como magic number el último bloque de la salida devuelta por el paso anterior (ignorando los bits de paridad). Dicho bloque es convertido a cadena para poder producir con ella el hash de la contraseña.

3. Obtener claves de Oracle

Dependiendo de la version que estemos utilizando de Oracle, las claves seran almacenadas en lugares diferentes. En nuestro caso hemos empleado la versión 10g, y la ubicación por defecto es la tabla llamada **DBA_USERS**.

El ataque puede ser enfocado de tres formas diferentes:

- **Prueba de caja negra**: Las pruebas realizadas mediante pentesting iran encaminadas a la obtención de un listado de usuarios y su respectivo hash que posteriormente procederemos a explotar mediante diferentes aplicaciones y técnicas. Este proceso constará de dos partes:
  - Intrusión al sistema, ya sea por cuentas con claves por defecto o bien utilizando algún exploit que aproveche alguna vulnerabilidad en el sistema.
  - Extracción de la relación usuarios y hashes, bien por haber obtenido una cuenta con privilegios, o por haber provocado escalada de privilegios. Aunque esto no ha de ser necesariamente así, si la configuración de seguridad no es adecuada, será suficiente con tener privilegios mínimos para acceder a tales datos.
- **Prueba de caja gris**: Bastará con obtener la relación de usuarios empleando credenciales no privilegiados que le hayan sido entregados. Como hemos comentado anteriormente, es frecuente que al existir una mala configuración de seguridad, los perfiles más bajos puedan ser comprometidos y obtener la relación de usuarios/hashes. La principal diferencia respecto a la prueba de caja negra, es que el auditor contará de partida con una cuenta en el sistema.
- **Pruebade caja blanca**: En este caso, el auditor solicitará al administrador que extraiga la relación de usuarios y hashes para analizarlas. Este será el caso que ejemplifiquemos.

Debemos de tener en cuenta que el atacante siempre tratará de acceder a las cuentas con más poder: **SYS**, **SYSTEM**, **DBSNMP**. Tampoco podemos menospreciar a los usuarios con privilegios de administrador. Si descuidamos un único perfil de estos, bastará con suplantar su identidad para tomar control total de la base de datos.

Para obtener el fichero con las relaciones de usuario/hash emplearemos el comando spool para que nos genere un fichero con la salida devuelta, para facilitarnos el trabajo y no tener que depender de una sesión de Oracle.

```
SQL> spool claves.txt
SQL> select username, password from dba_users;

USERNAME                       PASSWORD
------------------------------ ------------------------------
SYS                            B34603C7EA213830
SYSTEM                         DC595B547A74D7A5
ANONYMOUS                      anonymous
SEBAS                          8BEF3019900C8EBA
MDSYS                          72979A94BAD2AF80
OUTLN                          4A3BA55E08595C81
DIP                            CE4A36B8E06CA59C
TSMSYS                         3DF26A8B17D0F29F
FLOWS_FILES                    364B78B9EABB9E56

USERNAME                       PASSWORD
------------------------------ ------------------------------
CTXSYS                         D1D21CA56994CAB6
DBSNMP                         E066D214D5421CCC
FLOWS_020100                   16E4C012E98710D0
XDB                            E76A6BD999EF9FF1
HR                             4C6D73C3E8B0F0DA

16 rows selected.
SQL> spool off
```

4. Auditorizando las claves de Oracle

Existen diversas tecnicas para comprometer los hashes obtenidos. Asi mismo encontramos también un amplio abanico de herramientas para conseguir nuestro objetivo.

En nuestro caso, implementaremos los dos primeros tipos de ataques:

- **Fuerza bruta**: Es utilizado para recuperar una clave probando todas las combinaciones posibles hasta encontrar aquella que permita el acceso.

  Para ello se sirve del conocimiento del algoritmo de cifrado empleado y de un par texto claro/texto cifrado, realizando el cifrado (descifrado) de uno de los miembros del par con cada una de las posibles combinaciones de clave, hasta obtener el otro miembro del par.

  Un factor importante es el juego de caracteres utilizados, a menor número de caracteres utilizados, menores serán las combinaciones posibles. Sin embargo, dado que se basa en un método de prueba y error, su coste computacional será muy elevado cuando tratemos con claves de longitudes largas.

- **Ataque de diccionario**: Tecnica combinada con la fuerza bruta consistente en intentar averiguar una contraseña probando todas las palabras de un diccionario. Por norma general suelen ser mas eficientes, debido a que muchos usuarios y administradores de sistemas suelen utilizar una palabra existente en su lengua como contraseña, para recordar con mayor facilidad la clave.

  Sin embargo tienen un menor porcentaje de éxito contra sistemas que emplean contraseñas generadas al azar o robsutas, vease aquellas que combinan caracteres alfanumericos y símbolos.

- **Rainbow tables**: Son tablas de búsqueda que ofrecen una solución eficiente al eterno problema de tiempo/espacio en la recuperación de una clave cifrada mediante una función hash.

Antes de comenzar con los ataques, vamos a crearnos algunos usuarios adicionales con distintas claves:

```
SQL> CREATE USER prueba0 IDENTIFIED by qifmz;

User created.

SQL> CREATE USER prueba1 IDENTIFIED by aj09lz;

User created.

SQL> CREATE USER prueba2 IDENTIFIED by universidad;

User created.

SQL> CREATE USER prueba3 IDENTIFIED by auditoria;

User created.
```

4.1. Fuerza Bruta

Utilizaremos la herramienta **Orabf**, quizás una de las más populares y extendidas para aplicar este tipo de ataques. Su forma de uso es la siguiente:

```
Uso: orabf [hash]:[usuario] [opciones]

options:
-c [numero]     complejidad: numero de [1..6] o un fichero
   -            lee de la entrada estandar
   [fichero]    lee de un fichero
   1            numbers
   2            alpha
   3            alphanum
   4            standard oracle (alpha)(alpha,num,$\_$,$#$,\$)... (default)
   5            entire keyspace ($' '..'~'$)
   6            custom (charset read from first line of file: charset.orabf)
-m [numero]     max. longitud: Debe estar comprendida entre [1..14] (Def: 14)
-n [numero]     min. pwd longitud: Debe estar comprendida entre [1..14] (Def: 1)
-r              resumir: intenta volver a una sesion anterior
```

Esta herramienta permite pasarle un fichero de texto a modo de diccionario, pero en este caso sólamente utilizaremos las opciones de fuerza bruta.

Los resultados obtenidos para los distintos hashes han sido:

- **CEA979188ACD136D:PRUEBA0**
  39811147 passwords tried. password found: PRUEBA0:QIFMZ
  elapsed time 00:00:41
- **E9C53E30AD6A0CD7:PRUEBA1**
  110450911 passwords tried. password found: PRUEBA1:AJ09LZ
  elapsed time 00:01:58
- **59517CF5D6516C5B:PRUEBA2**
  465160255 passwords tried. current password: EGC_LB
  elapsed time 00:09:53 **RESULTA INVIABLE**
- **4F659BFB6FCCA81B:PRUEBA3**
  842261295 passwords tried. current password: WER1O3
  elapsed time 00:32:34 **RESULTA INVIABLE**

Un análisis más detallado, nos revelará el tiempo promedio usado para adivinar los hashes de claves con longitud variable:

- **319B5E833E3291F7:Z**
  27 passwords tried. password found: Z:Z
  elapsed time 00:00:00.
- **5A99821B918805AA:Z**
  703 passwords tried. password found: ZZ:Z
  elapsed time 00:00:00
- **748B146ADFA8EB5E:Z**
  18279 passwords tried. password found: ZZZ:Z
  elapsed time 00:00:00
- **05C69D5374090AF5:Z**
  475255 passwords tried. password found: ZZZZ:Z
  elapsed time 00:00:1

- **F29224D28E7B57EF:Z**
  12356631 passwords tried. password found: ZZZZZ:Z
  elapsed time 00:00:16
- **124F2EA043CD2ED1:Z**
  321272407 passwords tried. password found: ZZZZZZ:Z
  elapsed time 00:08:52
- **C717F96D746B3294:Z**
  8353082583 passwords tried. password found: ZZZZZZZ:Z
  elapsed time 03:16:55
- **7AAED8BB9D1B19F3:Z**
  217180147159 passwords tried. password found: ZZZZZZZZ:Z
  elapsed time 04d 09:12:38

Como se aprecia, se produce un consumo de tiempo considerabla a partir de claves cuya longitud sea superior a los seis caracteres. En estos casos, un ataque de fuerza bruta sería poco recomendable llevarlo a cabo, y deberíamos plantearnos utilizar otras técnicas como ataques de diccionario.

4.2. Ataque de diccionario

Para llevar acabo este tipo de ataque nos serviremos de un diccionario que contiene las principales palabras del castellano. El comando wc nos muestra que vamos a probar un total de 86193 claves diferentes:

```
sebas@sebas-laptop:~/roote/Hacking/Oracle\$ wc -l spanish.txt
86193 spanish.txt
```

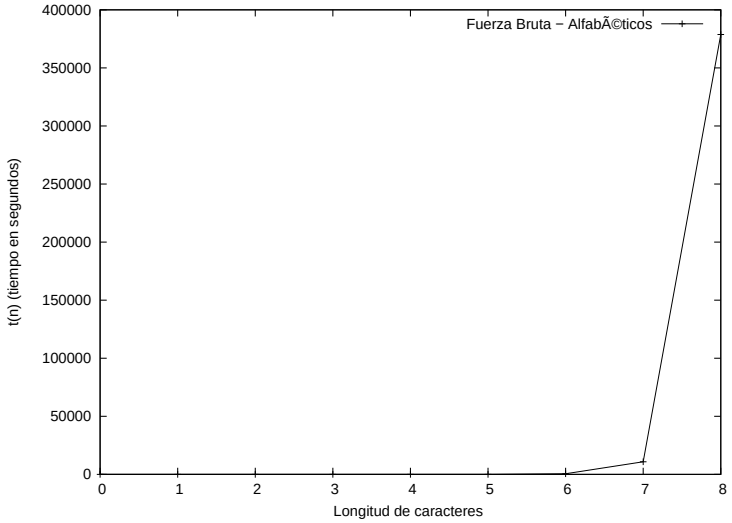
Utilizando los mismos valores hashes para el ataque de fuerza bruta, los resultados obtenidos para el ataque de diccionario han sido:

- **CEA979188ACD136D:PRUEBA0**
  password was not found in dictionary
  86193 passwords tried. elapsed time 00:00:00.
- **E9C53E30AD6A0CD7:PRUEBA1**
  password was not found in dictionary
  86193 passwords tried. elapsed time 00:00:00.
- **59517CF5D6516C5B:PRUEBA2**
  password found: PRUEBA2:UNIVERSIDAD
  83130 passwords tried. elapsed time 00:00:00.
- **4F659BFB6FCCA81B:PRUEBA3**
  password found: PRUEBA3:AUDITORIA
  10734 passwords tried. elapsed time 00:00:00.

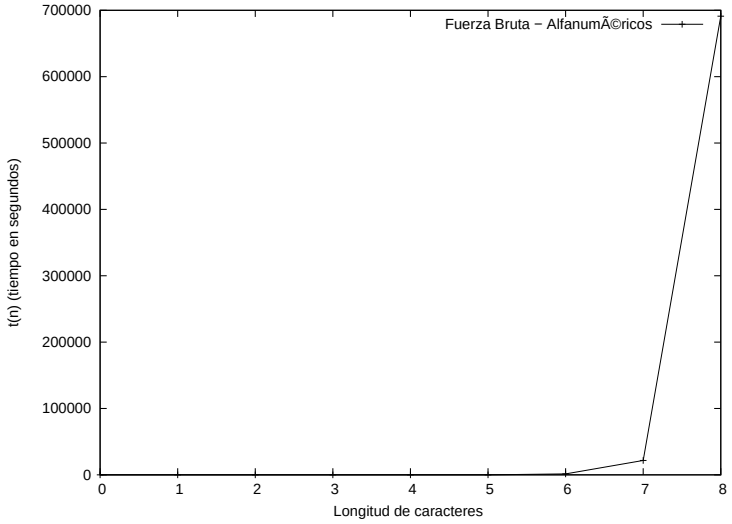
4.3. Conclusiones

Realizando diversas gráficas que recojan los resultados devueltos por nuestro ataque de fuerza bruta en un **Intel Core 2 Duo P8600 a 2.4Ghz** han sido:

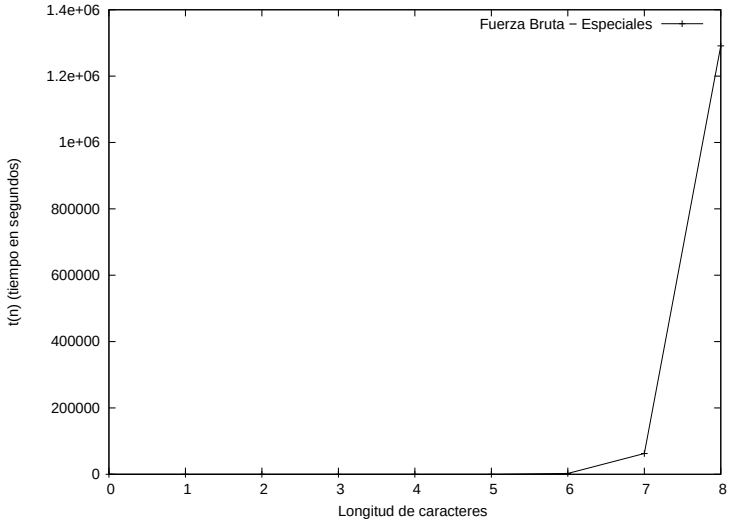
- Alfabeto de [0-8] caracteres - Expresión regular: Z+



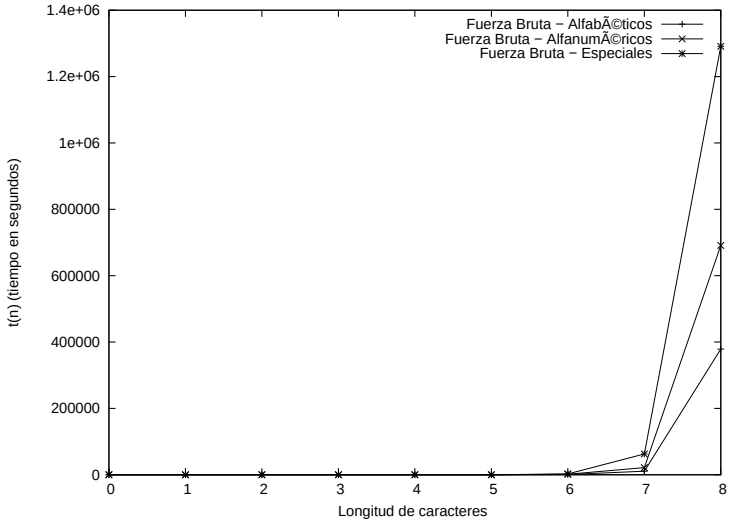
- Alfanumérico de [0-8] caracteres - Expresión regular: Z+[0-8]?



- Especial de [0-8] caracteres - Expresión regular: Z+[0-8]?[#_$]?



- Superposición casos anteriores



Como se puede apreciar en la comparativa, a mayor entropía, mayor tiempo de cálculo. Resultando inviable el ataque para contraseñas de longitud superior a 8 caracteres.

Para esos casos se ha demostrado que la mejor alternativa es usar un ataque de diccionario, siempre que tengamos la certeza de que el usuario ha utilizado alguna palabra que esté contenida en nuestro fichero. No obstante resultará inútil si la clave está compuesta por caracteres sin ningún tipo aparente de conexión entre ellos.

Otra posibilidad que cobra fuerza es el **Amazon Elastic Cloud Compute** (Amazon EC2), un servicio que permite a sus usuarios aunar la potencia de cálculo de varios procesadores localizados en distintos puntos geográficos. Permitiendo así reducir considerablemente el tiempo de cómputo.

Esto ha supuesto un peligro cuando hablamos del password cracking, ahora esos hashes que veíamos inviables vuelven a cobrar fuerza cuando hablamos del EC2. Una nueva puerta se abre, aunque todo tiene un precio, y no todos estamos al alcance de costearnoslo.[7]

5. Recomendaciones

Aquellas organizaciones que utilicen productos de Oracle para mantener sus base de datos pueden mitigar los aspectos negativos comentados previamente, imponiendo una politica de seleccion de contraseñas robustas, y obligando a cumplir ciertos minimos en cuanto a privilegios. De hecho las siguientes medidas permitirian a una empresa proteger su confidencialidad y los hashes de sus claves:

- Usar usuarios sin privilegios para aplicaciones web.
- Restringir el acceso a los hashes.
- Auditar las consultas SELECT en la vista DBA_USERS.
- Encriptar el trafico TNS.
- Forzar una longitud minima aceptable para las contraseñas.

Dichas recomendaciones se detallaran en mayor profundidad a continuacion.

5.1. Usar usuarios sin privilegios para aplicaciones web

Una de las técnicas para capturar los hashes de las contraseñas es explotar los permisos de los usuarios para la base de datos. Esto se realiza a través de ataques de inyección SQL.

Es recomendable utilizar una cuenta de usuario con los privilegios estrictamente necesario para ejecutar la aplicación. En ningún caso un usuario deberá ser miembro del grupo DBA.

5.2. Restringir el acceso a los hashes

Mientras que los hashes de Oracle son almacenados en la tabla SYS.USER$, el sistema crea por defecto varias vistas que pueden ser utilizadas para obtener información comprometida de la base de datos.

Un claro ejemplo es la vista DBA_TAB_COLS, que puede ser utilizada para identificar cualquier objeto de base de datos que incluya una columna llamada PASSWORD, mostramos un ejemplo

```
SQL> select owner, table_name from dba_tab_cols where column_name = 'PASSWORD';

OWNER                          TABLE_NAME
------------------------------ ------------------------------
SYS                            USER$
SYS                            USER_HISTORY$
SYS                            LINK$
SEBAS                          BIN$go8UG7a6zqngQAB/AQErTw==$0
SYS                            KU_USER_VIEW
SYS                            KU_PSW_HIST_LIST_VIEW
SYS                            KU_10_1_DBLINK_VIEW
SYS                            DBA_USERS
SYS                            EXU8PHS
SYS                            USER_DB_LINKS
SYS                            KU$_DBLINK_VIEW

OWNER                          TABLE_NAME
------------------------------ ------------------------------
SYS                            KU_ROLE_VIEW
SYS                            EXU8ROL

13 rows selected.
```

Una herramienta que se puede utilizar para ver los usuarios que tienen acceso a estos objetos, es **who_can_access.sql** desarrollada por Pete Finnigan, disponible en http://www.petefinnigan.co.uk/tools.htm.

5.3. Auditar las consultas SELECT en la vista DBA_USERS

Una opción es auditar la vista DBA_USERS utilizada para obtener los hashes de las contraseñas para las cuentas de la base de datos. Para activarlas bastará con:

```
SQL> audit select on dba_users;

Audit succeeded
```

Hay que tener en cuenta que un atacante podría hacer referencia a la tabla SYS.USER obteniendo directamente el hash sin hacer referencia a la vista DBA_USERS, evadiendo así la auditoría establecida anteriormente.
Intentar generar una auditoría sobre la tabla SYS.USER directamente, da el siguiente error:

```
SQL> audit select on sys.user$;
audit select on sys.user$

ERROR at line 1:
ORA-00701: object necessary for warmstarting database cannot be altered
```

Todavia no hay solucion a esto. Al menos en la version 9i y 10g.

5.4. Encriptar el tráfico TNS

Otra opcion a considerar seria encriptar el trafico TNS protegiendo informacion sensible, incluyendo los hashes de las contraseñas. Esto viene incluido por defecto en las opciones de **Seguridad Avanzada de Oracle** (OAS) en la version de Oracle Enterprise. No obstante no todo el mundo puede permitirse adquirir una licencia de este producto, asi que una opcion alternativa seria user SSH tunneling para cifrar todos los datos. OpenSSH (www.openssh.org) es un buen ejemplo de ello.

5.5. Forzar una longitud mínima aceptable para las contraseñas

Un buen administrador de sistemas puede implementar una función que verifique la entropía de una clave de usuario. Reduciendo las posibilidades de éxito de un ataque por fuerza bruta. Teniendo en cuenta que el tiempo estimado para romper una clave de 12 caracteres oscila en torno a los 60 días.

Esta función es muy simple, recibirá tres parámetros: *nombre de usuario*, *nueva contraseña* y *contraseña antigua*. Y comprobará que la longitud de la nueva contraseña sea superior a 12 caracteres, en caso de ser así realizará el cambio satisfactoriamente, de lo contrario devolverá un error. El código puede ser el siguiente:

```
CREATE OR REPLACE FUNCTION sys.password_verify
        (username varchar2, password varchar2, old_password varchar2)
        RETURN boolean IS
BEGIN
        IF length(password) < 12 THEN
                raise_application_error(-20000, 'Password less than 12 characters');
        END IF;
        RETURN(TRUE);
END;
```

6. Conclusion

El objetivo de este documento ha sido examinar el mecanismo utilizado por Oracle para proteger las contraseñas de sus usuarios. Hemos revisado el algoritmo utilizado para generar los distintos hashes y se han demostrado ciertas debilidades que permitirian a un atacante averiguar las contraseñas para un cierto usuario.

Se han explicado alternativas para forzar la obtención en texto plano de los hashes mediante fuerza bruta, y ataques de diccionario. Realizando una pequeña auditoría a algunos usuarios con sus respectivas claves que se han creado para ejemplizar este documento.

Por último se han puntualizado una serie de recomendaciones que cualquier administrador debería de seguir para aumentar la seguridad de su base de datos.

La finalidad que ha impulsado a escribir este documento ha sido en todo momento didáctica, y se desaconseja el uso de las herramientas presentadas en el mismo para impulsar cualquier tipo de actividad de dudosa legalidad.

7. Bibliografia

Referencias

[1] Robert Morris and Ken Thompson(1979, November)
Password Security: A Case History. Communications of the ACM

[2] Bob Baldwin. (1993, Julio 9)
Oracle Password Encryption Algorithm?
Usenet Newsgroup comp.databases.oracle
URL: http://groups-beta.google.com/group/comp.databases.oracle/msg/83ae557a977fb6ed?hl=en

[3] Pete Finnigan. (2005, Mayo)
"PeteFinnigan.com Tools, who_can_can_access script"
URL: http://www.petefinnigan.com/who_can_access.sql.

[4] vonJeek. - RevMoon. (2007, Marzo 25)
"The next level of Oracle attacks".

[5] David Litchfield. - Wiley ans Sons. (2007, January)
"The Oracle Hacker's Handbook".

[6] Sergio Hernando (2010, Febrero 27).
Fuerza bruta sobre claves Oracle.
URL: http://www.sahw.com/wp/archivos/2010/03/02/auditoria-de-contrasenas-en-oracle-database-3-de-4-fuerza-bruta-sobre-claves-oracle/.

[7] Electric Alchemy (2009, Octubre 29).
Cracking Passwords in the Cloud: Insights on Password Policies
URL: http://news.electricalchemy.net/2009/10/password-cracking-in-cloud-part-5.html